Carnegie Mellon
Software Engineering Institute

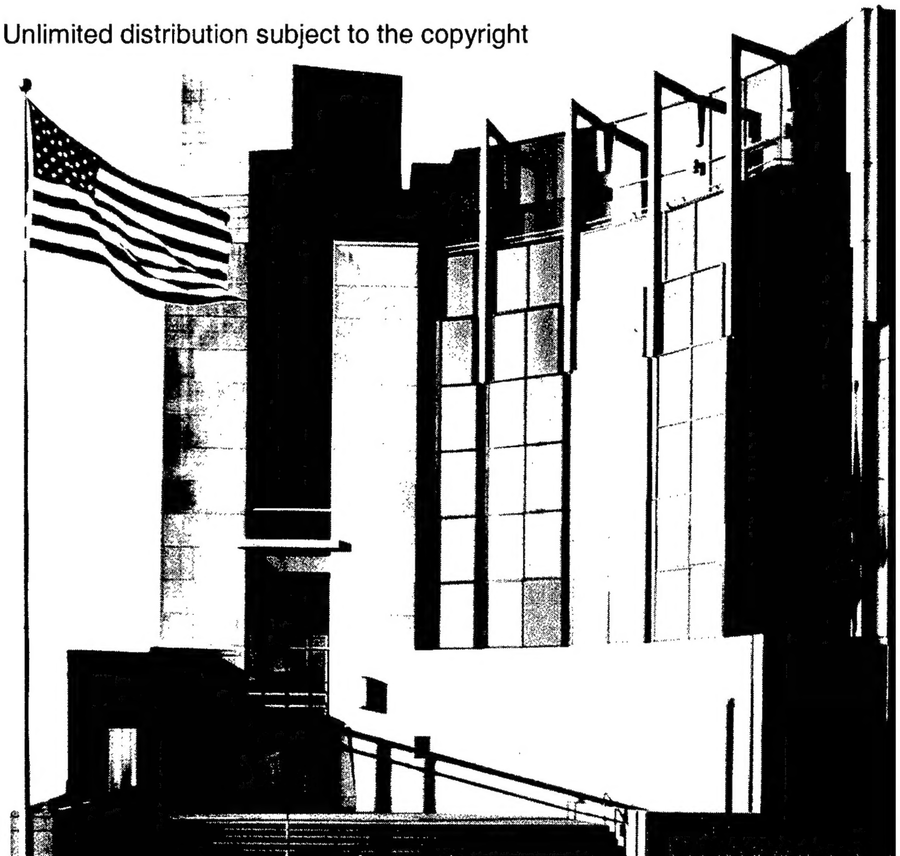
Model-Based Verification — Scope, Formalism, and Perspective Guidelines

David P. Gluch
Santiago Comella-Dorda
John Hudak
Grace Lewis
John Walker
Chuck Weinstock

October 2001

Performance Critical Systems

Technical Note
CMU/SEI-2001-TN-024



Model-Based Verification — Scope, Formalism, and Perspective Guidelines

David P. Gluch
Santiago Comella-Dorda
John Hudak
Grace Lewis
John Walker
Chuck Weinstock

October 2001

Performance Critical Systems

Technical Note
CMU/SEI-2001-TN-024

Contents

Abstract **v**

1 Model-Based Verification **1**

2 Introduction to Scope, Formalism, and Perspective **3**

2.1 Scope 4

2.2 Formalism 4

2.3 Perspective 6

2.4 The Broader Context of MBV 7

3 Generating the Statement of Scope, Formalism, and Perspective **8**

3.1 Team Process 8

3.2 Factors to Consider 9

3.3 Fine-Tuning the Scope, Formalism, and Perspective 10

4 Example of a Scope, Formalism, and Perspective Statement **12**

4.1 Template 12

4.2 Scope Section 12

4.3 Formalism Section 12

4.4 Perspective Section 13

4.5 Example of Scope, Formalism, and Perspective Statement 13

References **15**

List of Figures

Figure 1: Model-Based Verification Process and Artifacts 1

Abstract

The goal of model-based verification (MBV) is to reduce the number of defects. Like any other quality assurance (QA) technique, it is not equally efficient in every situation. It is critical to determine where and how to use MBV to achieve the largest impact in terms of the number and criticality of defects found with a reasonable amount of effort. This document provides guidance for defining the scope, formalism (approach and tools), and perspective for applying MBV. The critical (important or risky) aspects of the system and its development, including both programmatic and technical issues, drive these choices and form the basis for these guidelines.

1 Model-Based Verification

Model-based verification (MBV) is a systematic approach to finding defects (errors) in software requirements, designs, or code [Gluch 98]. The approach judiciously incorporates mathematical formalism, in the form of models, to provide a disciplined and logical analysis practice, rather than a "proof"-of-correctness strategy. MBV involves creating the essential models of system behavior and analyzing them against formal representations of the expected properties.

The artifacts and key processes used in MBV are shown in Figure 1. Model building and analysis are the core parts of MBV practices. These two activities are performed using an iterative and incremental approach, where a small amount of modeling is followed by a small amount of analysis. A parallel compile activity gathers detailed information on errors and potential corrective actions.

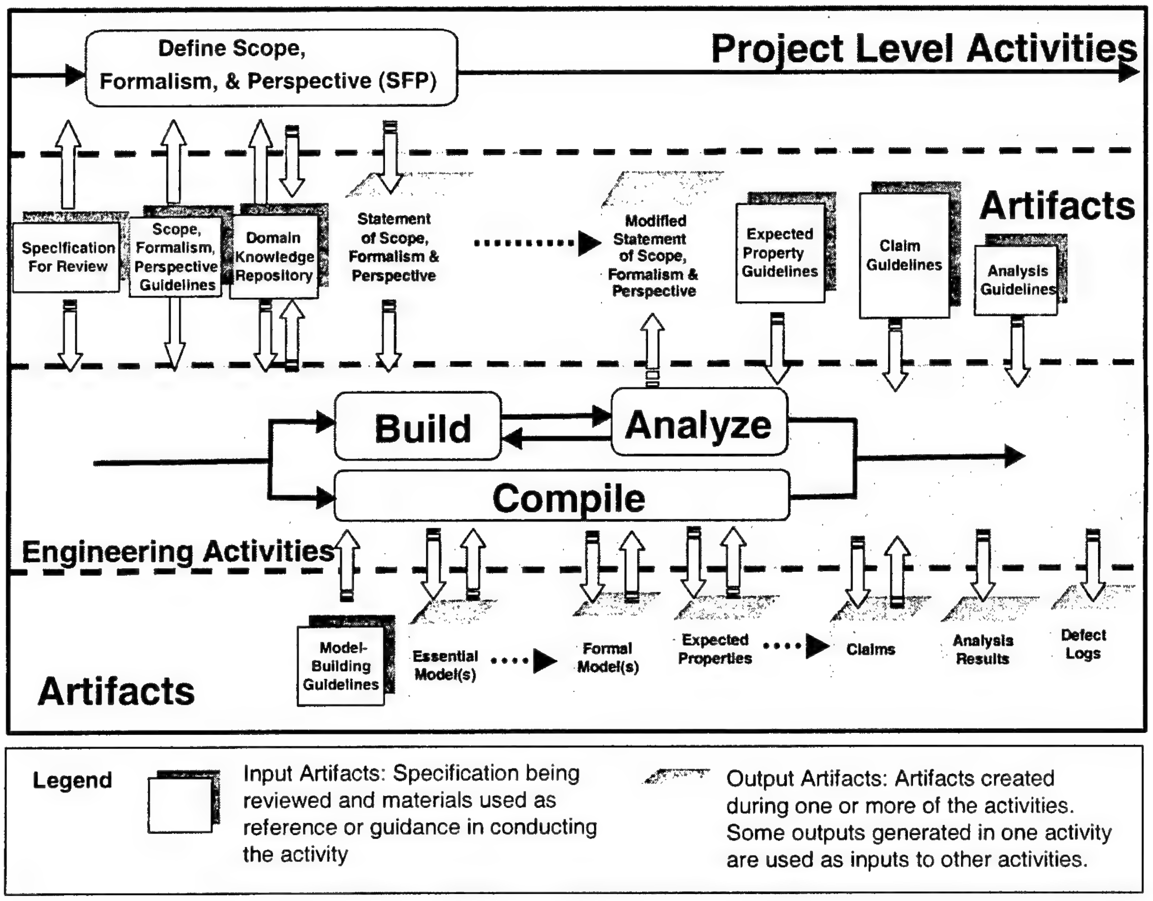

Define Scope, Formalism, & Perspective (SFP)
Project Level Activities
Artifacts
Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Statement of Scope, Formalism & Perspective
Modified Statement of Scope, Formalism & Perspective
Expected Property Guidelines
Claim Guidelines
Analysis Guidelines
Build
Analyze
Compile
Engineering Activities
Artifacts
Model-Building Guidelines
Essential Model(s)
Formal Model(s)
Expected Properties
Claims
Analysis Results
Defect Logs
Legend
Input Artifacts: Specification being reviewed and materials used as reference or guidance in conducting the activity
Output Artifacts: Artifacts created during one or more of the activities. Some outputs generated in one activity are used as inputs to other activities.

Figure 1: Model-Based Verification Process and Artifacts

Essential models are simplified formal representations that capture the essence of a system, rather than provide an exhaustive, detailed description of it. Through the selection of only critical (important or risky) parts of the system and appropriately abstracted perspectives, a reviewer using model-based techniques, can focus the analysis on the critical and technically difficult aspects of the system. Driven by the discipline and rigor required in the creation of a formal model, simply building the model, in and of itself, uncovers errors.

Once the formal model is built, it can be analyzed using automated model-checking tools such as SMV (Symbolic Model Verifier). Within this analysis, potential defects are identified while both formulating claims about the system's expected behavior and analyzing the model using automated model-checking tools. Model checking has been shown to uncover the especially difficult-to-identify errors: the kind of errors that result due to the complexity associated with multiple interacting and interdependent components [Clarke 96]. These include embedded as well as highly distributed applications.

Many different formal modeling and analysis techniques are employed within MBV [Gluch 98, Clarke 96]. The choices are based upon the type of system being analyzed and the technological foundation of the critical aspects of that system. Deciding which techniques to use involves an engineering tradeoff among the technical perspective, formalism, level of abstraction, and scope of the modeling effort.

The specific techniques and engineering practices of applying MBV to software verification have yet to be fully explored and documented. A number of barriers to the adoption of MBV have been identified including the lack of good tool support, expertise in organizations, good training materials, and process support for formal modeling and analysis.

In order to address some of these issues, the Software Engineering Institute (SEI) has created a process framework for model-based verification practices that identifies a number of key tasks and artifacts. Additionally, the SEI is working on a series of technical notes that can be used by MBV practitioners. Each technical note is focused on a particular MBV task, providing guidelines and techniques for one aspect of MBV practices. Currently, the technical notes that are planned address abstraction in building models, generating expected properties, generating formal claims, and interpreting the results of analysis.

This technical note provides guidance for defining the scope, formalism (approach and tools), and perspective for applying MBV. The critical (important or risky) aspects of the system and its development, including both programmatic and technical issues, drive these choices and form the basis for these guidelines.

2 Introduction to Scope, Formalism, and Perspective

In a world in which quality rules and resources are unlimited, we would probably verify every single subsystem from every possible angle and under the most stringent conditions. Unfortunately, resources are rarely unlimited, and quality is just another tradeoff in a complex management and engineering decision that also includes time to market, functionality, and economical viability.

In this imperfect real world, we need to focus our limited resources to have the largest impact with the smallest cost. MBV, as any other quality assurance (QA) technique, is not equally efficient in every situation. We have to decide where and how to use MBV to have the greatest impact in terms of the number and criticality of defects found. In particular, three parameters must be defined:

1. **scope**: The portion of a system that is to be modeled is its scope. For example, the scope could be to model all of the system at a high level of abstraction (e.g., model and analyze the fault response of the entire system). In other cases, the scope could be limited to model a specific part of the system in greater detail (e.g., model and analyze the communication module).
2. **formalism**: This parameter of the MBV effort defines the modeling approaches and tools to be used. Modeling techniques that can be employed in MBV include state-machine representations, process algebras, and rate-monotonic modeling. Each of those techniques is supported by a number of tools.
3. **perspective**: The modeling perspective is the context for representing the behavior and characteristics of a system. Perspective is the criteria and the guide for the abstraction process. A perspective could be the user's view of a system, or it could be the representation of a specific feature, function, or capability. For example, for an aircraft fly-by-wire computer system, one perspective would be to model the system from a fault-response perspective. The model might describe states of the system in terms of the number or type of faults that may occur (e.g., the no-fault state, single-fault state, and loss-of-roll attitude information). An alternative perspective might consider the flight modes of the system (e.g., takeoff, climb out, cruise, and landing).

For example, in reviewing the design specification for a networked system, the protocol response to network errors may be identified as critical. The protocol becomes the focus of the modeling effort (the scope), and a formal state-machine technique (formalism) is used to analyze its behavior in error situations (perspective).

Scope, formalism, and perspective are strongly dependent on each other. If, for example, we want to study the redundancy-management behavior of a system (perspective), we will surely focus on those subsystems that are redundant (scope), and use a behavior-modeling formalism (like state machines). Consequently, those decisions will be taken together because they rely on a common set of factors.

2.1 Scope

The scope delineates what will and will not be verified. The scope can be something as broad as the whole system or as narrow as a module running in one of many subsystems. The scope can be physical (a particular subsystem), logical (a module or sequence), or temporal (some time interval of the system operation). In some situations, it is useful to use different approaches to verify different parts of the system; this would correspond to a number of scopes associated to potentially different perspectives and formalisms. These are some examples of scope:

- the whole system
- a vital system component
- shared resources or shared data
- complex concurrent processes
- intricate switching logic
- complicated error sequences
- the interval of time from system started to system operational

Scope is mainly concerned with the decomposition of a large problem (the system) into more manageable subproblems. This is extremely important to make MBV viable as current practices do not scale well and can cope only with limited size problems. After model checking the different parts of the system, *compositional reasoning techniques* can be applied to infer global properties about the entire system [Berezin 98].

2.2 Formalism

Formalism defines the modeling technique that will be used to verify the system. During MBV, formalism is strongly determined by the selected scope and perspective; there are some formalisms that are better suited for a particular situation than others. If, for example, the temporal characteristics of a real-time system are being verified, we need a modeling formalism that supports temporal clauses. Other factors to consider include the in-house expertise and the depth required for the verification.

Knowledge of the various modeling techniques' capabilities is vital to making the right choice. In general, decisions on specific modeling techniques should consider the characteristics of the system being modeled, the efficacy of the modeling technique when applied to those characteristics, the complexity of the model being generated, and the risks associated with the system. In particular, the risks can help to determine the level of formality at which a system should be analyzed (e.g., a highly formalized model would be appropriate for a safety-critical system). High assurance often implies high cost, however, and these types of tradeoffs should be considered when choosing a modeling technique [Gluch 95]. The verification of the temporal-logic properties of finite-state systems has enjoyed significant growth and become the prominent formalism for the MBV of software systems [Clarke 86]. This prominence resulted from the fact that efficient algorithms have been developed that make it possible to verify digital systems with realistic size. Symbolic model checking, for example, can verify systems with more than 100 binary variables [Burch 90]. Additionally, this kind of model checking has shown remarkable success in verifying digital hardware designs and has shown some promising results in research studies on software-based systems.

A key issue in the verification of finite-state systems is the selection of the temporal language used to specify properties. Two possible views regarding the unfolding of time induce two types of temporal logic[1]. In *linear temporal logic* (LTL), the evolution of time is viewed as a sequence of states – a single line of possible states. In contrast, *computational tree logic* (CTL), a version of *branching temporal logic,* views the evolution of time as possibly proceeding along a multiplicity of paths. Each path is a linear sequence of states.

The last years have seen heated discussion about the relative merits of LTL and CTL [Vardi 01]. It has been argued that LTL is easier to use and that CTL is easier to verify and has better tool support. However, new developments are weakening these arguments. The complexity of CTL can be mitigated using tools and techniques that hide the details of the formalism[2] [Corbett 00a, Holt 99]. Also, the SMV and other tools currently support LTL. Personally, we have found both notations to be useful and worth exploring.

In addition to selecting a formalism, we need a tool that supports it. Most of the tools existing today are primarily research developments that require significant expertise on the part of users. However, commercial tools for software model checking are beginning to emerge, which are normally more user friendly and easier to use [I-Logix 01]. Examples of MBV-capable tools are

- Carnegie Mellon University's version of SMV [McMillan 92b]
- Bell Labs' SPIN [Holzmann 97]
- Kansas State University's Bandera [Corbett 00b]
- Microsoft's SLAM Toolkit [Thomas 01]
- University of Dortmund's VERDICT [Kowalewski 97]

1 Interval logic and other temporal logic variants are not covered in this report.

2 Comella-Dorda, Santiago; Gluch, Dave; Hudak, John; Lewis, Grace; & Weinstock, Chuck. *Model-Based Verification – Claim Creation Guidelines.* Pittsburgh, PA: Software Engineering Institute, to be published.

2.3 Perspective

The scope determines what areas of the system we are going to focus on, and the perspective states what is interesting about those areas. Normally, it is impractical to model the general behavior of a component as this behavior is often defined by an excessive number of states. A more specific view of the system component is recommended. For example, we can focus our effort on studying the control flow or the initialization sequence. The following is a list of possible perspectives for a particular component:

- interprocess communications
- flow of control
- redundancy management
- coherency (cache coherence)
- emergency procedures/failure modes
- initialization
- time-outs
- message sequencing – incorrect sequencing of events or messages

If the scope is characterized as a decomposition mechanism, the perspective can be seen as an abstraction mechanism. Deciding what to consider about a particular component is equivalent to deciding what to ignore. When we decide on a perspective, we are abstracting away all the characteristics and behaviors that are not related to the selected perspective.

The perspective guides not only the model-building activity, but also the expected property generation[1]. Expected properties are natural-language statements about the behavior of a system: behavior that is consistent with user expectations. Expected properties are formalized as claims and verified against system models. Building and verifying general statements about system behavior does not significantly improve confidence in the quality of the system, as these statements are necessarily too broad. A better approach is to be more focused and exhaustive and to gain confidence in some critical aspects of the system. Thus, the perspective provides not only the view for building models, but also the focus for defining expected properties.

Together the scope and perspective can be used to make the verification process iterative. Initially, a small component (scope) can be verified from a single perspective. Later, more components can be added, or more perspectives can be considered. This incremental approach can be followed until a satisfactory coverage of the subject system is reached. Since the scope and perspective will change over time, it may be interesting to put them under configuration management and track their evolution through the project.

[1] Gluch, Dave; Comella-Dorda, Santiago; Hudak, John; Lewis, Grace; & Weinstock, Chuck. *Guidelines for Generating Expected Properties*. Pittsburgh, PA: Software Engineering Institute, to be published.

2.4 The Broader Context of MBV

The broader context in which the MBV effort is implemented has a critical role in determining the scope, formalism, and perspective. Especially important is the extent to which MBV is integrated within the development effort.

Current software-development practices often involve a trial-and-error approach. Prototypes of the system are built; their feasibility is studied; and depending on the results of this analysis, a bigger prototype is built to study its feasibility. Translating this concept to civil engineering would mean building a small bridge, seeing if it falls down, and iteratively enlarging the toy-size bridge until the desired size is reached. This is unacceptable in civil engineering, but currently it is often the approach for building software systems.

We envision a paradigm in which software engineers manipulate models instead of real artifacts. The quality of the system is assured by the analysis and verification of models. In this context, MBV permeates the whole development effort. The project evolves through increasingly detailed models, and MBV practices ensure the correctness and quality of those models.

Currently, MBV is not well integrated into the development process. It can be seen as an add-on to improve software-engineering practices. Specifically, MBV is often applied as an augmentation of the peer-review activities [Gluch 99]. MBV improves a review team's ability to deal with complexity, by supplementing the "process formality" of facilitated group interactions with selective mathematical formality—formal modeling practices.

When MBV is used as an add-on to peer reviews, the definitions of scope, formalism, and perspective are included in the plan for the review. In particular, decisions must be made on what areas of the reviewed artifacts should be modeled, what modeling techniques to employ, and what expected properties are to be verified. That is, the scope, formalism, and perspective must be defined and documented. These choices, as well as considerations of the time and resources required for the review effort, will help the team coordinate modeling activities with those of the general review process.

In the following sections, we make the assumption that MBV is being used as an addition to development practices and not as an integral part of a model-centered development approach.

3 Generating the Statement of Scope, Formalism, and Perspective

The initial step in MBV is defining the objectives, approaches, and plans for execution of the MBV activity. The outcome of this planning step should include an initial statement of scope, formalism, and perspective, as it defines

- the aspects of the system that should be modeled
- the modeling techniques that are appropriate
- the properties of the system that must be verified

Critical aspects of the system should be used as the basis for these decisions. The amount of risk involved as well as the importance of relevant requirements determines which aspects are critical. Since generally it is not feasible to model all of the system in detail, the choice of these critical aspects requires substantial domain knowledge as well as knowledge about the relevant implementation details of the system. To be effective in these decision processes, it is imperative that the team defining the plan has a broad understanding of the system's requirements, design, and environment. If this is not the case, an individual with this knowledge should be included in the planning activities.

3.1 Team Process

Making decisions about the scope, formalisms, and perspective as a team exercise is important, as different people have different insights about the problem. The process of obtaining a consensus helps to guarantee that every important factor has been considered. Additionally, the people who are involved in such a decision may become more committed to implementing the decision that is eventually made. The latter is especially important if those who decide on the scope, formalism, and perspective will also be conducting the MBV (which is something that we strongly recommend).

The literature listed in the References section of this document discusses several group-decision-making techniques including Delphi [Linstone 75] and the Analytic Hierarchy Process (AHP) [Saaty 80]. These techniques are all well known and extensively documented, so we are not going to cover them in this report. In fact, the particular technique is not as important as ensuring that the following goals are achieved in the decision-making process:

- The decision must benefit from the expertise of every team member. This goal may be threatened by the presence of members with strong personalities and a vested interest in making their opinion prevail over their peers'.

- The team interaction has to provide additional insight, different from that of the team members. The group has to be more than the sum of its members. The team leader has to foster an environment in which there are no "wrong" inputs and every comment initiates constructive discussion.
- Every assumption has to be validated by the team as a whole. Often, this will reveal any weaknesses and shortcomings in people's assumptions. Team members have to be critical and refuse to accept anything *a priori*.
- The final decision has to be fully endorsed by every member of the team.

In order to achieve these goals, the team requires a diverse set of backgrounds including system and domain experts, MBV practitioners, and QA advocates.

The following section enumerates some factors and recommendations to be considered while discussing scope, formalism, and perspective.

3.2 Factors to Consider

There are a variety of factors, both programmatic and technical, to consider when establishing the scope, formalism, and perspective. These factors are also important considerations in the generation of expected properties and claims for the system.

We are defining programmatic factors loosely as those that are interesting from a management point of view and take into consideration nontechnical issues. Budget and schedule are probably the best well-known programmatic factors. For example, if there are x number of dollars to spend in MBV activities, the team should not select a scope that requires $10*x$ dollars.

Economical analysis and return on investment (ROI) are important factors to consider when defining the scope, formalism, and perspective. Every QA technique has the same basic constraint: defects are removed at a cost. Out of all the QA activities, software inspections have proved to be one of the most cost-effective techniques, because the early detection of defects results in a large savings [Ebenau 94, Fagan 76]. (The cost of fixing defects increases geometrically with the project phase [Basili 01]). MBV is a more systematic and formal approach to inspections. While this makes MBV more expensive than traditional reading-based inspections, it also makes MBV more suitable for finding certain kinds of defects. Given those characteristics, we can make a cost/benefit assessment based on the following factors:

- the costs associated with applying MBV
- the number, criticality, and complexity of those defects likely to be found by MBV that might not be found in a traditional inspection
- the side benefits derived from MBV including a better understanding of the system behavior and the potential later use of the models created during the MBV effort
- the cost of failure due to a defect that was not found in time

This analysis can help to determine the areas and aspects of the system in which it makes economical sense to conduct MBV. It can also determine the extent and level of formalism at which MBV should be applied.

There is another family of non-quantifiable programmatic factors that are concerned with politics, expectation management, and customer relations. For example, there may be a special interest in a particular component not because of its criticality or technical complexity, but because the customer showed special concern about it. This type of programmatic factor may appear arbitrary, but it is critical for the long-term success of a project.

Technical factors, on the other hand, are the day-to-day concern of engineers. These factors cover the viability of the system in terms of its quality attributes including correctness, performance, reliability, and others. Engineers should determine what is critical for the system operation, what is technically challenging, and what is better suited for MBV. The following are a few examples of technical concerns that may determine the scope, formalism, and perspective:

- unprecedented systems or parts of systems
- complex resource contention
- intense real-time/time-driven systems
- high reliability
- concurrency or distributed functions
- extensive interactions among components

The careful consideration of these and related factors will increase the chances of selecting a scope, formalism, and perspective that leverages the capabilities of MBV and makes sensible use of available resources.

3.3 Fine-Tuning the Scope, Formalism, and Perspective

Independently of the method used to decide the scope, formalism, and perspective, the decision must be revisited. Every software project has peculiarities that make it impossible to select the optimal combination of scope, formalism, and perspective *a priori*. As the verification effort progresses, practitioners will get a better sense of what areas are benefiting most from the modeling activities. The team will discover areas with poor quality or unexpected combinational complexity and find out which formalism fits better considering the current system and the level of effort that is required.

The system environment is surely going to evolve, and this evolution will make many of the elements of the scope, formalism, and perspective obsolete after a while. This dynamic nature of the system makes a tight collaboration between the MBV reviewers and the system analysts/developers critical. A good way of ensuring this tight collaboration is to present the results from the model-checking effort to the development team. This presentation will validate the defects found in addition to providing valuable feedback for refining the scope, formalism, and perspective statement.

4 Example of a Scope, Formalism, and Perspective Statement

The Scope, Formalism, and Perspective Statement is a project-level document that guides MBV activities and is usually modified as a result of the iterative nature of the process, especially in the Perspective section that will evolve as the modeling is performed. The Scope and Formalism sections are more stable because they are more global. They generally require a team decision for modification.

4.1 Template

The Scope, Formalism, and Perspective Statement is a short document based on the following template.

1. Scope
    - 1.1. Description
    - 1.2. Rationale
2. Formalism
3. Perspective
    - 3.1. General Description
    - 3.2. Specific Guidelines
    - 3.3. Issues Requiring Special Consideration
    - 3.4. System Attributes Explicitly Ignored

4.2 Scope Section

The *Description* describes the system or subsystems that will be modeled. The *Rationale* is a short paragraph that explains why this area is being explored and the reasons for building the model.

4.3 Formalism Section

This section identifies the approaches that will be used to model the system, often summarizes how and to what portion of the system the tools might be applied, and sometimes suggests other approaches that may be used. The Formalism section may address specific techniques, tools, notations, and configuration-management issues.

4.4 Perspective Section

The *General Description* describes the focus of the effort and includes

1. the properties to investigate (e.g., tolerance to a specific set of faults, state completeness, and message protocol completeness)
2. the subsystems involved as they relate to the scope that is established for the analysis effort (e.g., fire-control computer, interfaces to radar, and rocket's weapon subsystem)
3. operational modes or profiles (scenarios) (e.g., automatic acquisition and targeting or making a specific variation of a chemical)

The *Specific Guidelines* are statements about critical aspects and issues. They should identify specific considerations and provide guidance for investigating them. They can also be seen as precursors to or explicit statements of expected properties. These statements may include some or all of the following:

1. initial conditions/assumptions about external entities
2. operational sequences or modes of operation
3. outputs and associated attributes (persistent, transient, periodic, etc.)
4. validation of "normal" modes of operation (the intended operation)
5. specific abnormal operations and behavior (Certain abnormal conditions should be handled in a specific way.)
6. timing issues (e.g., synchronization, deadlock, and livelock)
7. protocol/algorithmic validation (if checking protocols or algorithms, the issues that are important)

The *Issues Requiring Special Consideration* section is a short, descriptive paragraph of each issue that is of special concern and should be explored with greater attention. This should include an explanation of the rationale for highlighting this issue (i.e., why this area is being explored and why this model is being built).

Finally, the *System Attributes Explicitly Ignored* section states any issues that don't need to be addressed and the rationale for excluding them.

4.5 Example of Scope, Formalism, and Perspective Statement

1. Scope

1.1. Description

The MBV activities will focus on the unpiloted aircraft's flight-control computer system, the communications network that connects the sensors, the flight-control computer, and the actuators.

1.2. Rationale

These systems are part of the flight and safety-critical elements of the aircraft.

2. Formalism

State-machine modeling and the SMV approach will be used as well as possibly SPIN for the analysis of the communications protocol.

Other approaches and tools may be applied as needed. These will be assessed based upon the results of the analysis effort. Changes will be made as appropriate through the normal project tracking and planning processes.

3. Perspective

3.1. General Description

The modeling will focus on the redundancy-management aspects of the system and the associated fault responses. The total set of operational modes and all possible fault scenarios should be considered for the key subsystems identified in the scope.

3.2. Specific Guidelines

- Consider only the basic processing states of each redundant component.
- Consider the normal and fault responses of the checker.
- Consider the coordination of outputs sent to the checker.
- Explore the fault response logic in the checker.
- Address the impact of potential error states identified by the checker.

3.3. Issues Requiring Special Consideration

The synchronization strategy is not defined explicitly in the specification. The implications of this should be explored in detail.

3.4. System Attributes Explicitly Ignored

The algorithms used by the flight-data-processing units are ignored explicitly, because they are being analyzed by another verification team.

References

[Basili 01]	Basili, Vic & Boehm, Barry. "Software Defect Reduction Top 10 List." *Computer 34*, 1 (January 2001): 135-137.
[Berezin 98]	Berezin, S.; Campos, S.; & Clarke, E. *Compositional Reasoning in Model Checking* (CMU-CS-98-106). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University, February 1998. Available WWW: <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-106.ps> (1998).
[Burch 90]	Burch, J.R.; Clarke, E.M.; McMillan, K.L.; Dill, D.L.; & Hwang, L.J. "Symbolic Model Checking: 10^{20} States and Beyond," 428-439. *Proceedings of the 5th Symposium on Logic in Computer Science.* Philadelphia, PA, June 4-7, 1990. Los Alamitos, CA: IEEE Computer Society Press, 1990.
[Campos 94]	Campos, S.; Clarke, E.; Marrero, W.; Minea, M.; & Hiraishi, H. "Computing the Quantitative Characteristics of Finite-State Real-Time Systems," 266-270. *Proceedings of the 1994 IEEE Real-Time Systems Symposium.* San Juan, Puerto Rico, December 7-9, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994
[Clarke 86]	Clarke, E.M.; Emerson, E.A.; & Sistla, A.P. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications." *ACM Transactions on Programming Languages and Systems 8*, 2 (January 1986): 244–263.
[Clarke 94]	Clarke, E.; Grumberg, O.; & Hamaguchi, K. "Another Look at LTL Model Checking," 415-424. *Proceedings of the Sixth International Conference on Computer-Aided Verification CAV '94.* Stanford, CA, June 21-23, 1994. Lecture Notes in Computer Science, Volume 818. New York, NY: Springer-Verlag, 1994.
[Clarke 95]	Clarke, Edmund M., et al. "Verification of the Futurebus+ Cache Coherence Protocol." *Formal Methods in System Design 6*, 2 (March 1995): 217-232.

[Clarke 96] Clarke, E.M. & Wing, Jeannette. "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys 28*, 4 (December 1996): 626-643.

[Corbett 00a] Corbett, James C.; Dwyer, Matthew B.; Hatcli, John; & Robby. "A Language Framework For Expressing Checkable Properties of Dynamic Software." *Proceedings of the 7th SPIN Workshop*. Stanford, CA, August 30 - September 1, 2000. Lecture Notes in Computer Science Volume 1885, ISSN 0302-9743. New York, NY: Springer-Verlag, September 2000.

[Corbett 00b] Corbett, James; Dwyer, Matthew; Hatcliff, John; Pasareanu, Corina; Robby; Laubach, Shawn; & Zheng, Hongjun. "Bandera: Extracting Finite-State Models from Java Source Code." *Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Ireland, June 4-11, 2000. Los Alamitos, CA: IEEE Computer Society Press, 2000.

[Ebenau 94] Ebenau, Robert G. & Strauss, Susan H. *Software Inspection Process*. New York, NY: McGraw-Hill, 1994.

[Fagan 76] Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal 15*, 3 (1976): 182-211.

[Gluch 95] Gluch, David P.; Dorofee, Audrey J.; Hubbard, Elizabeth A.; & Travalent, John J. *A Collaboration in Implementing Team Risk Management* (CMU/SEI-95-TR-016, ADA309157). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995. Available WWW: <http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.016.html> (1995).

[Gluch 98] Gluch, D. & Weinstock, C. *Model-Based Verification: A Technology for Dependable System Upgrade* (CMU/SEI-98-TR-009, ADA354756). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW <http://www.sei.cmu.edu/publications/documents/98.reports/98tr009/98tr009abstract.html> (1998).

[Gluch 99]	Gluch, David P. & Brockway, Jared. *An Introduction to Software Engineering Practices Using Model-Based Verification* (CMU/SEI-99-TR-005, ADA366089). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, April 1999. Available WWW: <http://www.sei.cmu.edu/publications/documents/99.reports/ 99tr005/ 99tr005abstract.html> (1999).
[Holt 99]	Holt, Alexander. "Formal Verification with Natural Language Specifications: Guidelines, Experiments and Lessons So Far." *South African Computer Journal* 24 (November 1999): 253-257.
[Holzmann 97]	Holzmann, Gerard J. "The Model Checker SPIN." *IEEE Transactions on Software Engineering 23*, 5 (May 1997): 279-295.
[I-Logix 01]	I-Logix, Inc. "I-Logix Introduces Formal Verification Technology to Systems Engineering with the Release of the Statemate MAGNUM Model Checking Products" [online]. Available WWW: <URL: http://www.ilogix.com/frame_html.cfm> (May 16, 2001).
[Kowalewski 97]	Kowalewski, Stefan & Treseler, Heinz. "VERDICT - A Tool for Model-Based Verification of Real-Time Logic Process Controllers," 217-221. *Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems.* Geneva, Switzerland, April 1-3, 1997. Los Alamitos, CA: IEEE Computer Society Press, 1997.
[Lichtenstein 85]	Lichtenstein, O. & Pnueli, A. "Checking that Finite-State Concurrent Programs Satisfy Their Linear Specification," 97-107. *Conference Record of the Twelfth ACM Symposium on the Principles of Programming Languages.* New Orleans, Louisiana, January 13-16, 1985. New York, NY: ACM, 1985.
[Linstone 75]	Linstone, H.A. & Turoff, M. *The Delphi Method: Techniques and Application.* Reading, MA: Addison-Wesley, 1975.
[McMillan 92a]	McMillan, K.L. *Symbolic Model Checking: An Approach to the State Explosion Problem* (CMU-CS-92-131). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University, 1992.
[McMillan 92b]	McMillan, K.L. "Symbolic Model Checking - An Approach to the State Explosion Problem." PhD diss., School of Computer Science, Carnegie Mellon University, 1992.

[Saaty 80]	Saaty, T.L. *The Analytic Hierarchy Process*. New York, NY: McGraw-Hill, 1980.
[Thomas 01]	Ball, Thomas & Rajamani, Sriram K. "Automatically Validating Temporal Safety Properties of Interfaces," 103-122. *Proceedings of the 8th International SPIN Workshop on Model Checking of Software*. Toronto, Canada, May 19-20, 2001. Lecture Notes on Computer Science, Volume 2057. New York, NY: Springer-Verlag, 2001.
[Vardi 01]	Vardi, Moshe Y. "Branching Vs. Linear Time: Final Showdown," 1-22. *Proceedings of TACAS 2001 - Tools and Algorithms for the Construction and Analysis of Systems*. Genova, Italy, April 2-6, 2001. Lecture Notes in Computer Science, Volume 2031. New York, NY: Springer-Verlag, 2001.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	3. REPORT DATE October 2001	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Model-Based Verification — Scope, Formalism, and Perspective Guidelines		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) David P. Gluch, Santiago Comella-Dorda, John Hudak, Grace Lewis, John Walker, Chuck Weinstock		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2001-TN-024
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The goal of model-based verification (MBV) is to reduce the number of defects. Like any other quality assurance (QA) technique, it is not equally efficient in every situation. It is critical to determine where and how to use MBV to achieve the largest impact in terms of the number and criticality of defects found with a reasonable amount of effort. This document provides guidance for defining the scope, formalism (approach and tools), and perspective for applying MBV. The critical (important or risky) aspects of the system and its development, including both programmatic and technical issues, drive these choices and form the basis for these guidelines.

14. SUBJECT TERMS model-based verification, MBV, scope, formalism, perspective	15. NUMBER OF PAGES 26
16. PRICE CODE	

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102